

Enforcing Opacity in Modular Systems

Graeme Zinck* Laurie Ricker* Hervé Marchand**
Loïc Hélouët**

* Mount Allison University, Canada
(e-mail: {gbzinck, lricker}@mta.ca)

** Inria Rennes-Bretagne Atlantique, France
(e-mail: firstname.lastname@inria.fr)

Abstract: Opacity of a secret in a discrete system is a property that guarantees that some behaviors or states can never be inferred with certainty from partial observation of the system. Enforcing opacity in a discrete-event system, encoded by a finite labelled transition system (LTS), is a way to avoid information leakage. Checking opacity is decidable but costly (EXPTIME in the worst cases). This paper addresses opacity for modular systems in which every module, represented by an LTS, has to protect its own secret (a set of secret states \mathcal{S}) w.r.t. a local attacker. Once the system composed, we assume a coalition between the attackers that share their local view (called the global attacker). Assuming the global attacker can observe all interactions between modules, we provide a reduced-complexity opacity verification technique and an algorithm for constructing local controllers that enforce opacity for each secret separately.

Keywords: opacity, supervisory control, labelled transition systems, modular systems.

1. INTRODUCTION

Online services are an increasingly important part of the global economy. Since we depend on them for financial transactions, confidential communications, and administrative processes, we need to enforce security and privacy. In addition to protecting stored data, it is also important to guarantee secrecy of the tasks performed online. Indeed, launching an attack at an appropriate moment may dramatically increase the chances of success of the attack; for instance, a phishing attack may be more effective when a user is about to complete a transaction. Hence, during a transaction, a system should remain as obscure as possible to actors that are not involved in the exchange.

One way of evaluating security in systems using formal methods is the property of opacity. Opacity was first introduced in the context of modelling security protocols (Mazaré, 2005). The notion was later studied with respect to transition systems, in which a predicate is considered opaque if an observer of the system cannot deduce the predicate's truth with any certainty (Bryans et al., 2008). In this way, opacity unifies two security properties— anonymity and secrecy—to evaluate information flow. It evaluates whether an observer can infer a “secret” about the system behavior, assuming the observer knows the system's structure and has partial observation of the system's actions. If the observer's estimate of the system's behavior never reveals the secret, the system is opaque. There are several variants of opacity: for instance, *initial-state opacity* ensures that an attacker can never distinguish when the system starts at a secret initial state, and *current-state opacity* ensures that an attacker can never distinguish when the system is currently in a secret state. See (Jacob et al., 2016) for an review of these variants.

This paper considers opacity in a modular architecture. Modular systems are typically modeled using parallel com-

position of transition systems, which can then be used for diagnosis and control applications; e.g., (de Queiroz and Cury, 2000; Contant et al., 2006). By assembling modules, the resulting global transition system may be very large. Thus, a key issue when working with modular systems is to avoid computing the global system; we want to verify and enforce properties only on relevant sets of components. While there are contexts where all components are relevant, this approach can significantly improve the complexity of the proposed solutions. Saboori and Hadjicostis (2010) have leveraged this modular property to verify initial-state opacity with reduced complexity. In this paper, we apply similar techniques to verify current-state opacity in systems where each module has an individual secret to protect. When a modular system is not opaque, we want to enforce opacity using supervisory control, i.e., restrict behaviors of the system to ensure its opacity. The works of (Dubreil et al., 2008, 2010) introduced the idea of applying supervisory control theory to enforce opacity. We use these techniques to construct local controllers that enforce opacity of each module's secret in composed systems. That is, we find modular controllers that collectively ensure the global system does not leak individual secrets.

The remainder of this paper is organized as follows. First, we introduce our notation and background on discrete event systems, opacity, and supervisory control. Then, we present a reduced-complexity algorithm for verifying opacity in modular systems, assuming the attacker can observe the interface between modules. With this algorithm, we show that we do not always need to construct the monolithic global system to verify opacity. Finally, we present an algorithm for synthesizing local controllers that enforce opacity in modular systems.

2. PRELIMINARIES

2.1 Languages and Labelled Transition Systems

Let Σ be an alphabet of events. Then, Σ^* denotes the set of all finite words composed of elements in Σ . We can define a language as $\mathcal{L} \subseteq \Sigma^*$. A *word* is a sequence of events $t \in \Sigma^*$, which has length $|t|$. The *concatenation* of $t_1 = \sigma_1 \cdots \sigma_{|t_1|}$ and $t_2 = \sigma'_1 \cdots \sigma'_{|t_2|}$ is defined as $t_1.t_2 = \sigma_1 \cdots \sigma_{|t_1|}\sigma'_1 \cdots \sigma'_{|t_2|}$. The *prefix-closure* of a language \mathcal{L} is defined as $\bar{\mathcal{L}} = \{t \in \Sigma^* \mid (\exists t' \in \Sigma^*) t.t' \in \mathcal{L}\}$. Given an alphabet $\Sigma' \subseteq \Sigma$, we define the *projection operator* on finite words, $P_{\Sigma'} : \Sigma^* \rightarrow \Sigma'^*$, that removes from a word of Σ^* all the events that do not belong to Σ' . Formally, $P_{\Sigma'}$ is recursively defined as follows: $P_{\Sigma'}(\varepsilon) = \varepsilon$ and for $u \in \Sigma^*, \sigma \in \Sigma$, $P_{\Sigma'}(u.\sigma) = P_{\Sigma'}(u).\sigma$ if $\sigma \in \Sigma'$, or $P_{\Sigma'}(u)$ otherwise. Let $\mathcal{L} \subseteq \Sigma^*$ be a language. The definition of projection for words extends to languages: $P_{\Sigma'}(\mathcal{L}) = \{P_{\Sigma'}(s) \mid s \in \mathcal{L}\}$. Conversely, let $\mathcal{L} \subseteq \Sigma'^*$. The *inverse projection* of \mathcal{L} is $P_{\Sigma'}^{-1}(\mathcal{L}) = \{s \in \Sigma^* \mid P_{\Sigma'}(s) \in \mathcal{L}\}$.

Our model is a deterministic *labelled transition system* (LTS). An LTS G is defined as a four-tuple $G = (\mathcal{Q}, \Sigma, \delta, q_0)$, where \mathcal{Q} is a finite set of *states*, Σ is the alphabet of events, $\delta : \mathcal{Q} \times \Sigma \rightarrow \mathcal{Q}$ is a *partial transition map*, and $q_0 \in \mathcal{Q}$ is the *initial state*.

The transition map easily extends to words: for a state $q \in \mathcal{Q}$, a word $t \in \Sigma^*$, where $|t| \geq 1$, and event $\sigma \in \Sigma$, we can recursively define $\delta(q, t.\sigma)$ as $\delta(q, t.\sigma) = \delta(\delta(q, t), \sigma)$. We will write $\delta(q, \sigma)!$ (resp., $\delta(q, t)!$) when action σ is *allowed* from state q , i.e., $\exists q' \in \mathcal{Q}, \delta(q, \sigma) = q'$ (resp., when sequence t is allowed from q).

Let $\mathcal{L}(G)$ denote the language of the LTS G ; that is, $\mathcal{L}(G) = \{t \in \Sigma^* \mid \delta(q_0, t)!\}$. By definition, this language is prefix-closed: $\mathcal{L}(G) = \bar{\mathcal{L}}(G)$. We denote the language of G with respect to a set of states $\mathcal{X} \subseteq \mathcal{Q}$ as $\mathcal{L}_{\mathcal{X}}(G)$, where $(\forall t \in \Sigma^*) \delta(q_0, t) \in \mathcal{X} \Leftrightarrow t \in \mathcal{L}_{\mathcal{X}}(G)$.

The notion of a projection extends to an LTS. To show the partial observation of a system G , we construct the *determinization* of G . In this structure, only observable events Σ_o are used as transitions. Each state in the structure represents the set of states in which the original system could be after observing a word in Σ_o^* . More formally, we define the *unobservable reach* of a state $q \in \mathcal{Q}$, with respect to an alphabet $\Sigma_o \subseteq \Sigma$, as $\text{UR}_{\Sigma_o}(q) = \{q' \in \mathcal{Q} \mid (\exists t \in (\Sigma \setminus \Sigma_o)^*) \delta(q, t) = q'\}$. Then, we define the *determinization* of G with respect to Σ_o as $\text{Det}_{\Sigma_o}(G) = (\mathcal{X}, \Sigma_o, \delta_{\Sigma_o}, x_0)$, where $\mathcal{X} = 2^{\mathcal{Q}}$, $x_0 = \text{UR}_{\Sigma_o}(q_0)$, and, for $x \in \mathcal{X}$ and $\sigma \in \Sigma_o$, $\delta_{\Sigma_o}(x, \sigma) = \bigcup_{q \in x} \text{UR}_{\Sigma_o}(\delta(q, \sigma))$. Thus, the set of observable words is:

$$P_{\Sigma_o}(\mathcal{L}(G)) = \mathcal{L}(\text{Det}_{\Sigma_o}(G)). \quad (1)$$

In this paper, we work with systems designed by composing n modules, described by transition systems G_1, G_2, \dots, G_n , that synchronize on common events. We can build a single monolithic system by composing all modules. We define the *parallel composition* operation for both languages and LTSs, which are used interchangeably in proofs. Given two languages $\mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$, we define their parallel composition as $\mathcal{L}_1 \parallel \mathcal{L}_2 = P_{\Sigma_2}^{-1}(\mathcal{L}_1) \cap P_{\Sigma_1}^{-1}(\mathcal{L}_2)$. From this, we make two observations:

Remark 1. For languages $\mathcal{L}_1 \in \Sigma_1^*, \mathcal{L}_2 \in \Sigma_2^*, t \in \mathcal{L}_1 \parallel \mathcal{L}_2 \Leftrightarrow P_{\Sigma_1}(t) \in \mathcal{L}_1 \wedge P_{\Sigma_2}(t) \in \mathcal{L}_2$. As a consequence, parallel composition is commutative ($\mathcal{L}_1 \parallel \mathcal{L}_2 = \mathcal{L}_2 \parallel \mathcal{L}_1$) and associative ($(\mathcal{L}_1 \parallel \mathcal{L}_2) \parallel \mathcal{L}_3 = \mathcal{L}_1 \parallel (\mathcal{L}_2 \parallel \mathcal{L}_3)$).

Remark 2. Given $t \in \mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$, then $\{t\} \parallel \mathcal{L}_2 \subseteq \mathcal{L}_1 \parallel \mathcal{L}_2$.

We define the parallel composition of two LTSs, $G_i = (\mathcal{Q}_i, \Sigma_i, \delta_i, q_{0,i})$ for $i \in \{1, 2\}$, by $G_1 \parallel G_2 = (\mathcal{Q}_1 \times \mathcal{Q}_2, \Sigma_1 \cup \Sigma_2, \delta_1 \times \delta_2, (q_{0,1}, q_{0,2}))$, where $\delta_1 \times \delta_2$ is defined as follows:

$$\delta_1 \times \delta_2((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)), & \text{if } \delta_1(q_1, \sigma)! \wedge \delta_2(q_2, \sigma)!; \\ (\delta_1(q_1, \sigma), q_2), & \text{if } \delta_1(q_1, \sigma)! \wedge \sigma \notin \Sigma_2; \\ (q_1, \delta_2(q_2, \sigma)), & \text{if } \sigma \notin \Sigma_1 \wedge \delta_2(q_2, \sigma)!; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Performing the parallel composition on languages and on LTSs is equivalent: that is,

$$\mathcal{L}(G_1 \parallel G_2) = \mathcal{L}(G_1) \parallel \mathcal{L}(G_2). \quad (2)$$

The *interface* between two modules G_i and G_j , where $i, j \in \{1, \dots, n\}$ and $i \neq j$, is the shared alphabet $\Sigma_{i,j} = \Sigma_i \cap \Sigma_j$. The two LTSs synchronize on these events during the parallel composition, modelling joint events between the modules. Furthermore, for modules G_1, \dots, G_n , we denote the interface of shared events by:

$$\Sigma_s = \bigcup_{i,j \in \{1, \dots, n\}, i \neq j} \Sigma_{i,j}$$

Example 3. When we compose the two LTSs in Figures 1(a) and 1(b), we get $G_1 \parallel G_2$ as pictured in Figure 1(c). Note that the parallel composition prevents event c at state $2'$ in G_2 because the two LTSs must synchronize on the shared events in $\Sigma_{1,2} = \{c\}$. \square

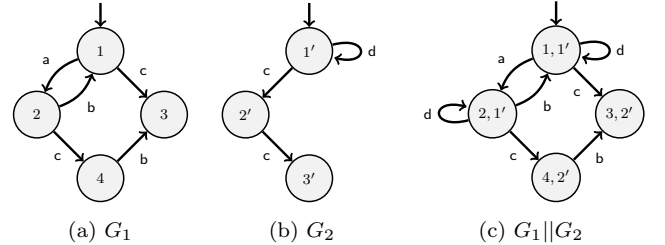


Fig. 1. LTSs G_1, G_2 , and $G_1 \parallel G_2$ with $\Sigma_s = \{c\}$

2.2 Opacity

To model security properties in systems, we use the concept of *current-state opacity*. For a system G , we assume the presence of an attacker that has access to an alphabet $\Sigma_a \subseteq \Sigma$. Furthermore, the system has a set of secret states $\mathcal{S} \subseteq \mathcal{Q}$. The *secret language* of a system is the set of words that end in a secret state: $\mathcal{L}_{\mathcal{S}}(G) = \{t \in \mathcal{L}(G) \mid \delta(q_0, t) \in \mathcal{S}\}$. Since our transition functions are deterministic, every sequence $t \in \mathcal{L}(G)$ is either in $\mathcal{L}_{\mathcal{S}}(G)$ or in $\mathcal{L}(G) \setminus \mathcal{L}_{\mathcal{S}}(G)$, where \setminus represents set subtraction. Let us recall a general definition of opacity from Cassez et al. (2012), where \mathcal{L} and $\mathcal{L}_{\mathcal{S}}$ need not be regular.

Definition 4. For a language $\mathcal{L} \subseteq \Sigma^*$ and secret language $\mathcal{L}_{\mathcal{S}} \subseteq \mathcal{L}$, $\mathcal{L}_{\mathcal{S}}$ is *opaque* with respect to \mathcal{L} and Σ_a iff $\forall t_1 \in \mathcal{L}_{\mathcal{S}}, \exists t_2 \in \mathcal{L} \setminus \mathcal{L}_{\mathcal{S}}$ where $P_{\Sigma_a}(t_1) = P_{\Sigma_a}(t_2)$.

Applying this to an LTS G with a set of secret states $\mathcal{S} \subseteq \mathcal{Q}$, \mathcal{S} is opaque with respect to G and Σ_a when $\mathcal{L}_{\mathcal{S}}(G)$ is opaque with respect to $\mathcal{L}(G)$ and Σ_a . Checking this condition amounts to verifying a reachability property of the determinization of G w.r.t. Σ_a . More formally, we have the following:

Proposition 5. (Follows from Definition 4). Let $Det_{\Sigma_a}(G) = (\mathcal{X}, \Sigma_a, \delta_{\Sigma_a}, x_0)$. \mathcal{S} is opaque with respect to G and Σ_a if and only if $\forall x \in \mathcal{X}, \forall t \in \Sigma_a^$ such that $x = \delta_{\Sigma_a}(\{x_0\}, t)$, $x \setminus \mathcal{S} \neq \emptyset$.*

This implies a simple algorithm for opacity verification for a secret \mathcal{S} w.r.t. an LTS G and attacker alphabet Σ_a :

Algorithm 1 (Cassez et al., 2012). Verify opacity of \mathcal{S} with respect to G and Σ_a .

```

1: function ISOPAQUE( $G, \Sigma_a$ )
2:    $Det_{\Sigma_a}(G) = (\mathcal{X}, \Sigma_a, \delta_{\Sigma_a}, x_0)$ 
3:   if  $(\exists t \in \Sigma_a^*, \delta_{\Sigma_a}(x_0, t) \setminus \mathcal{S} = \emptyset)$  then
4:     return false
5:   else
6:     return true

```

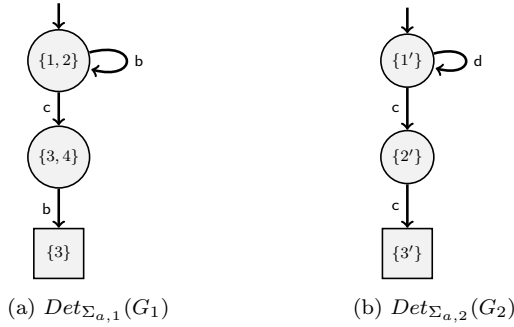


Fig. 2. LTSs $Det_{\Sigma_{a,1}}(G_1)$ and $Det_{\Sigma_{a,2}}(G_2)$ with $\Sigma_{a,1} = \{b, c\}$ and $\Sigma_{a,2} = \{c, d\}$

Example 6. Using the LTSs from Figure 1, suppose that in G_1 , the set of secret states is $\mathcal{S}_1 = \{3\}$, and in G_2 , $\mathcal{S}_2 = \{3'\}$. To verify if \mathcal{S}_i is opaque, we determinize G_i with respect to $\Sigma_{a,i}$, as pictured in Figure 2. Since the state $\{3\}$ is accessible from the initial state of $Det_{\Sigma_{a,1}}(G_1)$, and since $\{3\} \setminus \mathcal{S}_1 = \emptyset$, \mathcal{S}_1 is not opaque with respect to G_1 and $\Sigma_{a,1}$. Similarly, \mathcal{S}_2 is not opaque with respect to G_2 and $\Sigma_{a,2}$. \square

In Cassez et al. (2012), Algorithm 1 has been proven to be PSPACE-complete. Indeed, one can select a subset x of \mathcal{S} in polynomial time and then check whether $\{q_0\}$ is co-reachable from x in PSPACE. The hardness part comes from an easy reduction from a language inclusion problem. Hence, opacity verification is in EXPTIME.

2.3 Supervisory Control for Opacity

When opacity does not hold, we want to restrict the system's behavior to ensure secrets are not revealed. Using techniques from supervisory control, we can synthesize a controller to enforce opacity. We partition events in Σ into events that the controller can observe, Σ_o , and those that it cannot observe, Σ_{uo} , such that $\Sigma = \Sigma_o \uplus \Sigma_{uo}$. We also partition Σ into events the controller can control, Σ_c ,

and those it cannot control, Σ_{uc} , such that $\Sigma = \Sigma_c \uplus \Sigma_{uc}$. We assume that $\Sigma_c \subseteq \Sigma_o$.

The goal of a controller is to enforce a desired subset of behavior $K \subseteq \mathcal{L} \subseteq \Sigma^*$ by enabling or disabling events in Σ_c . Below, we recall two properties of a controller's language.

*Definition 7. [From Ramadge and Wonham (1982)]. A language $K \subseteq \mathcal{L}$ is **controllable** with respect to \mathcal{L} and Σ_c iff $\overline{K} \cdot (\Sigma \setminus \Sigma_c) \cap \mathcal{L} \subseteq \overline{K}$.*

*Definition 8. [From Lin and Wonham (1988)]. A language $K \subseteq \mathcal{L}$ is **observable** with respect to \mathcal{L} and Σ_o iff $\overline{K} = P_{\Sigma_o}^{-1}(P_{\Sigma_o}(\overline{K})) \cap \mathcal{L}$.*

Formally, we define a controller for G as a function $C : \mathcal{L}(Det_{\Sigma_o}(G)) \rightarrow \{\gamma \in 2^\Sigma \mid \Sigma_{uc} \subseteq \gamma\}$ and as an LTS $\mathcal{C} = (Q_{\mathcal{C}}, \Sigma_o, \delta_{\mathcal{C}}, q_{0,\mathcal{C}})$ such that for each $t \in \mathcal{L}(Det_{\Sigma_o}(G))$, $\delta_{\mathcal{C}}(q_{0,\mathcal{C}}, t.\sigma) \neq \emptyset \iff \sigma \in C(t)$. A controller C is *fully permissive* iff $C(t) = \Sigma$ for every $t \in \Sigma_o^*$. The controlled system denoted by \mathcal{C}/G has the language $\mathcal{L}(\mathcal{C}/G)$. The controller is *valid* if $\mathcal{L}(\mathcal{C}/G)$ is both controllable and observable with respect to $\mathcal{L}(G)$, Σ_o , and Σ_c . Since \mathcal{C} has the alphabet Σ_o , it is inherently observable, so it suffices to simply ensure that it is controllable.

There exists a largest sublanguage K of $\mathcal{L}(G)$ that does not violate the opacity of \mathcal{S} . When $\Sigma_c \subseteq \Sigma_o$, there exists a supremal controllable and observable prefix-closed sublanguage K^\dagger of $K \subseteq \mathcal{L}(G)$, which is the largest regular language included in K that can be enforced by control (Dubreil et al., 2010). If $K^\dagger \neq \emptyset$, then there exists a *supremal controller* \mathcal{C} such that $\mathcal{L}(\mathcal{C}/G) = K^\dagger$.

To compute a regular supremal controller, there are two possible assumptions about the attacker's knowledge: either the attacker has full knowledge of both G and \mathcal{C} 's construction, or the attacker only has knowledge of G 's construction. In the former case, Dubreil et al. (2010) proposed an algorithm to synthesize the supremal controller. This algorithm has a running time in $\mathcal{O}(|\mathcal{Q}| \times 2^{|\mathcal{Q}|})$ under two assumptions: first, $\Sigma_c \subseteq \Sigma_o$, and second, Σ_o is comparable to Σ_a (that is, $\Sigma_o \subseteq \Sigma_a$ or $\Sigma_a \subseteq \Sigma_o$). In the latter case, where an attacker only knows G , Tong et al. (2018) proposed a synthesis algorithm with running time in $\mathcal{O}(2^{2(|\mathcal{Q}| \times 2^{|\mathcal{Q}|} + |\Sigma_c|)})$, and only assumes $\Sigma_c \subseteq \Sigma_o$. In this paper, we mainly focus on this latter case, and do not assume that the observable and controllable alphabets are comparable.

3. VERIFYING OPACITY IN MODULAR SYSTEMS

The techniques for verifying opacity in a single system can be extended to systems composed of multiple modules. First, we can get the global system by performing the parallel composition between the modules. With this, we can verify the opacity of the secrets from all modules. However, the global system may be large and expensive to construct. In this section, we find optimizations that allow us to avoid constructing the entire monolithic system in some cases.

3.1 Problem Formulation

We are interested in systems composed of multiple modules, G_1, \dots, G_n , where the resulting system $G_1 \parallel \dots \parallel G_n$

has an alphabet $\Sigma = \bigcup \Sigma_i$. In such a system, we want to make sure the secrets within each module are kept confidential. We assume that we have n attackers, $\mathcal{A}_1, \dots, \mathcal{A}_n$, each observing one of the modules with the alphabet $\Sigma_{a,i} \subseteq \Sigma_i$ such that if $\sigma \in \Sigma_{a,j}$ and $\sigma \in \Sigma_i$, then $\sigma \in \Sigma_{a,i}$. That is, an event that is observable in one module is observable in all modules. Each attacker is trying to identify when its module is in a secret state of \mathcal{S}_i . We are interested in the case where all of these attackers share knowledge; the coalition of attackers can effectively observe events in $\Sigma_a = \bigcup \Sigma_{a,i}$. For simplicity, we refer to the attackers' collective view of the system as \mathcal{A} .

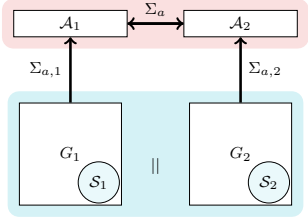


Fig. 3. Modular system architecture with attackers that share knowledge

We want to prevent \mathcal{A} from identifying with certainty that G_i is in a secret state in \mathcal{S}_i . Since we are working with the attackers' collective view, we need to ensure that every secret $(\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque in the global system $G_1 || \dots || G_n$ with respect to the combined attacker alphabet, Σ_a . Since we want to ensure this for each secret \mathcal{S}_i , $i \in \{1, \dots, n\}$, this leads to the following problem we intend to solve:

Problem 9. Given n modules G_i with secrets \mathcal{S}_i , and given a global attacker view \mathcal{A} with alphabet Σ_a , can we verify that $\mathcal{S} = \bigcup_{i=1}^n (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque with respect to $G_1 || \dots || G_n$ and Σ_a ?

As mentioned in Section 2.2, this has a simple solution: we can determinize the full system $G_1 || \dots || G_n$ with respect to the alphabet Σ_a and check for states that are a subset of \mathcal{S} . If we let $m = \max_{i \in \{1, \dots, n\}} |Q_i|$, then this algorithm is $\mathcal{O}(2^{m^n})$ because we first construct a system from n systems of size m and then determinize this result. However, it would be desirable to verify opacity without constructing this monolithic system, whenever possible.

In this paper, we assume that the modules' shared alphabet is a subset of the attacker's alphabet: $\Sigma_s \subseteq \Sigma_a$. That is, the attacker observes the interface between modules, acting as middleware in a network. This is the same assumption made in Saboori and Hadjicostis (2010) for verifying initial-state opacity. Under this assumption, we present optimizations for verifying current-state opacity.

3.2 Opacity and Composition

When we assume $\Sigma_s \subseteq \Sigma_a$, we can avoid determinizing the monolithic system by instead determinizing the local modules individually. This reduces the complexity of verifying opacity, making it feasible to perform verification on large modular systems. The following theorem and corollary show that this is possible.

Theorem 10. [From de Queiroz and Cury (2000)]. Let $\mathcal{L}_1 \subseteq \Sigma_1^*$ and $\mathcal{L}_2 \subseteq \Sigma_2^*$ be languages and let $\Sigma_s \subseteq \Sigma' \subseteq \Sigma_1 \cup \Sigma_2$. Then, $P_{\Sigma'}(\mathcal{L}_1 || \mathcal{L}_2) = P_{\Sigma' \cap \Sigma_1}(\mathcal{L}_1) || P_{\Sigma' \cap \Sigma_2}(\mathcal{L}_2)$.

Corollary 11. Let G_1, G_2 be LTSs and let $\Sigma_s \subseteq \Sigma' \subseteq \Sigma_1 \cup \Sigma_2$. Then, $\mathcal{L}(Det_{\Sigma' \cap \Sigma_1}(G_1) || Det_{\Sigma' \cap \Sigma_2}(G_2)) = \mathcal{L}(Det_{\Sigma'}(G_1 || G_2))$.

Proof.

$$\begin{aligned} & \mathcal{L}(Det_{\Sigma' \cap \Sigma_1}(G_1) || Det_{\Sigma' \cap \Sigma_2}(G_2)) \\ &= P_{\Sigma' \cap \Sigma_1}(\mathcal{L}(G_1)) || P_{\Sigma' \cap \Sigma_2}(\mathcal{L}(G_2)), \quad \text{By (1).} \\ &= P_{\Sigma'}(\mathcal{L}(G_1) || \mathcal{L}(G_2)), \quad \text{By Theorem 10.} \\ &= P_{\Sigma'}(\mathcal{L}(G_1 || G_2)), \quad \text{By (2).} \\ &= \mathcal{L}(Det_{\Sigma'}(G_1 || G_2)), \quad \text{By (1).} \end{aligned}$$

Thus, when computing what attackers can observe in the system, we can build $\mathcal{A} = Det_{\Sigma_{a,1}}(G_1) || \dots || Det_{\Sigma_{a,n}}(G_n)$ rather than $Det_{\Sigma_a}(G_1 || \dots || G_n)$. In this system, the state set is $\mathcal{X} = 2^{\mathcal{Q}_1} \times \dots \times 2^{\mathcal{Q}_n}$ while the secret state set is $\mathcal{S} = \bigcup_{i=1}^n (2^{\mathcal{Q}_1} \times \dots \times 2^{\mathcal{Q}_{i-1}}) \times 2^{\mathcal{S}_i} \times (2^{\mathcal{Q}_{i+1}} \times \dots \times 2^{\mathcal{Q}_n})$.

Following from Proposition 5, we can formalize when one of these individual secrets \mathcal{S}_i is opaque with respect to the composed system and attacker's alphabet:

Theorem 12. Let $G_i = (\mathcal{Q}_i, \Sigma_i, \delta_i, q_{0,i})$ with $\mathcal{S}_i \subseteq \mathcal{Q}_i$. Let the attacker's view of the system be $Det_{\Sigma_a \cap \Sigma_1}(G_1) || \dots || Det_{\Sigma_a \cap \Sigma_n}(G_n) = (\mathcal{X}, \Sigma_a, \delta_{\Sigma_a}, x_0)$ and assume $\Sigma_s \subseteq \Sigma_a$. \mathcal{S}_i is opaque with respect to the composed system $G_1 || \dots || G_n$ and Σ_a iff $\forall x \in \mathcal{X}, \forall t \in \Sigma_a^*$ such that $x = \delta_{\Sigma_a}(x_0, t)$, $x \notin (2^{\mathcal{Q}_1} \times \dots \times 2^{\mathcal{Q}_{i-1}}) \times 2^{\mathcal{S}_i} \times (2^{\mathcal{Q}_{i+1}} \times \dots \times 2^{\mathcal{Q}_n})$.

Proof. Since parallel composition is commutative and associative, we can prove that the secret \mathcal{S}_1 of module G_1 is opaque, without loss of generality. Let $Det_{\Sigma_a}(G_1 || \dots || G_n) = (\mathcal{X}', \Sigma_a, \delta'_{\Sigma_a}, x'_0)$. By definition, \mathcal{S}_1 is opaque with respect to $G_1 || \dots || G_n$ and Σ_a iff $\forall x \in \mathcal{X}', \forall t \in \Sigma_a^*$ such that $x = \delta'_{\Sigma_a}(\{x'_0\}, t)$, $x \setminus \mathcal{S}_1 \times \mathcal{Q}_2 \times \dots \times \mathcal{Q}_n \neq \emptyset$ —in other words, when $x \notin 2^{\mathcal{S}_1 \times \mathcal{Q}_2 \times \dots \times \mathcal{Q}_n}$. Thus, it is opaque when $\mathcal{L}_{2^{\mathcal{S}_1 \times \mathcal{Q}_2 \times \dots \times \mathcal{Q}_n}}(Det_{\Sigma_a}(G_1 || \dots || G_n)) = \emptyset$. By Corollary 11, $\mathcal{L}_{2^{\mathcal{S}_1 \times \mathcal{Q}_2 \times \dots \times \mathcal{Q}_n}}(Det_{\Sigma_a \cap \Sigma_1}(G_1) || \dots || Det_{\Sigma_a \cap \Sigma_2}(G_n)) = \mathcal{L}_{2^{\mathcal{S}_1 \times \mathcal{Q}_2 \times \dots \times \mathcal{Q}_n}}(Det_{\Sigma_a}(G_1 || \dots || G_n)) = \emptyset$. ■

By stating that \mathcal{S}_i is opaque with respect to $G_1 || \dots || G_n$ and Σ_a , we equivalently mean $\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1} \times \mathcal{S}_i \times \mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n$ is opaque with respect to $G_1 || \dots || G_n$ and Σ_a .

Corollary 13. Let $G_i = (\mathcal{Q}_i, \Sigma_i, \delta_i, q_{0,i})$ with $\mathcal{S}_i \subseteq \mathcal{Q}_i$. If every \mathcal{S}_i is opaque with respect to $G_1 || \dots || G_n$ and Σ_a , and $\Sigma_s \subseteq \Sigma_a$, then $\mathcal{S} = \bigcup_{i=1}^n (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque with respect to $G_1 || \dots || G_n$ and Σ_a .

Proof. Let the attacker's view of the system be $\mathcal{A} = (\mathcal{X}, \Sigma_a, \delta_{\mathcal{A}}, q_{0,\mathcal{A}})$. Since every \mathcal{S}_i is opaque with respect to $G_1 || \dots || G_n$ and Σ_a , we know $(\forall x \in \mathcal{X}) x \notin (2^{\mathcal{Q}_1} \times \dots \times 2^{\mathcal{Q}_{i-1}}) \times 2^{\mathcal{S}_i} \times (2^{\mathcal{Q}_{i+1}} \times \dots \times 2^{\mathcal{Q}_n})$ by Theorem 12. Thus, $(\forall x \in \mathcal{X}) x \notin \bigcup_{i=1}^n (2^{\mathcal{Q}_1} \times \dots \times 2^{\mathcal{Q}_{i-1}}) \times 2^{\mathcal{S}_i} \times (2^{\mathcal{Q}_{i+1}} \times \dots \times 2^{\mathcal{Q}_n})$. Thus, \mathcal{S} is opaque with respect to $G_1 || \dots || G_n$ and Σ_a . ■

This gives a compositional opacity verification algorithm with a reduced complexity: we can determinize all module, compose them, and verify opacity of secret states.

Theorem 14. When $\Sigma_s \subseteq \Sigma_a$, opacity verification of modular systems can be performed in $\mathcal{O}(2^{mn})$.

Proof. Theorem 12 shows that opacity of a secret can be checked on the composition of determinized modules. Let $m = \max_{i \in \{1, \dots, n\}} |Q_i|$. First, each module is determinized, giving at most 2^m states per module. Then, all n modules are composed and we check for opacity. Thus, the complexity is $\mathcal{O}(2^{mn})$. ■

This is exponentially better than the previous algorithm, but this algorithm can still be improved.

In general, a secret \mathcal{S}_i that is opaque with respect to G_i and Σ_a need not be opaque with respect to $G_1 || \dots || G_n$ and Σ_a . In other words, opacity of secrets is not preserved by composition. Note that synchronization on common events may forbid behaviors in a component, and thus reduce ambiguity as to whether a subsystem is in a secret state or not. Furthermore, synchronizing with another component may produce new observations, and hence reveal that G_i is in a state of \mathcal{S}_i . Nevertheless, there are clearly exceptions, such as when $\Sigma_s = \emptyset$.

Remark 15. Assuming $\Sigma_s = \emptyset$, for all $i \in \{1, 2\}$, \mathcal{S}_i is opaque with respect to G_i and Σ_a iff \mathcal{S}_i is opaque with respect to $G_1 || G_2$ and Σ_a .

We can use Remark 15 to split up a modular opacity verification problem into multiple subproblems to reduce the complexity. That is, we can partition the modules into sets that have no shared events with modules in other sets. Jezequel and Fabre (2012) use an *interaction graph* to illustrate this. The graph is defined by vertices, which are the LTS modules, and edges, where G_i has an edge to G_j if and only if $\Sigma_{i,j} \neq \emptyset$. We partition the LTSs into connected components, where the i -th connected component is called M_i . We can independently verify opacity for each connected component of modules; opacity of a secret \mathcal{S}_i holds for the whole system iff it holds for the parallel composition of the LTSs in a connected component, by Remark 15.

Example 16. Suppose we have four modules G_1, G_2, G_3, G_4 . Furthermore, suppose $\Sigma_{1,3}, \Sigma_{1,4}, \Sigma_{2,3}, \Sigma_{2,4} = \emptyset$ and $\Sigma_{1,2}, \Sigma_{3,4} \neq \emptyset$. Then, we yield the interaction graph in Figure 4 with two connected components: M_1 and M_2 . We can verify opacity independently within each connected component M_i . □

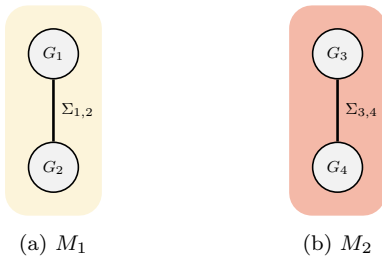


Fig. 4. Interaction graph of four modules

We can treat each connected component as a separate modular system and perform the algorithms on them separately. Throughout the rest of this paper, we assume that G_1, \dots, G_n are all part of the same connected component.

Furthermore, similar to observations from Saboori and Hadjicostis (2010) with respect to initial state opacity, current-state opacity holds under composition when the shared alphabet $\Sigma_s \subseteq \Sigma_a$. The following lemma and theorem are similar to those from Saboori and Hadjicostis (2010), but address current-state opacity.

Lemma 17. Let $\mathcal{L}_1 \subseteq \Sigma_1^*$, $\mathcal{L}_2 \subseteq \Sigma_2^*$, and $\Sigma_s \subseteq \Sigma_a$. Suppose that $t_1, t'_1 \in \mathcal{L}_1$ such that $P_{\Sigma_{a,1}}(t_1) = P_{\Sigma_{a,1}}(t'_1)$, and suppose that $t_2 \in \mathcal{L}_2$. Then, $\exists t \in \{t_1\} || \{t_2\}$ iff $\exists t' \in \{t'_1\} || \{t_2\}$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$.

Proof. (\Rightarrow) Suppose that $\exists t \in \{t_1\} || \{t_2\}$.

$$\begin{aligned} & P_{\Sigma_a}(\{t_1\} || \{t_2\}) \\ &= P_{\Sigma_{a,1}}(\{t_1\}) || P_{\Sigma_{a,2}}(\{t_2\}), \quad \text{by Theorem 10.} \\ &= P_{\Sigma_{a,1}}(\{t'_1\}) || P_{\Sigma_{a,2}}(\{t_2\}), \quad \text{since } P_{\Sigma_{a,1}}(t_1) = P_{\Sigma_{a,1}}(t'_1). \\ &= P_{\Sigma_a}(\{t'_1\} || \{t_2\}), \quad \text{by Theorem 10.} \end{aligned}$$

Since the two languages $\{t_1\} || \{t_2\}$ and $\{t'_1\} || \{t_2\}$ are equal after projecting onto Σ_a , there must exist a $t' \in \{t'_1\} || \{t_2\}$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$.

(\Leftarrow) Now, suppose $\exists t' \in \{t'_1\} || \{t_2\}$. By the same reasoning, $\exists t \in \{t_1\} || \{t_2\}$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$. ■

Theorem 18. For $i \in \{1, 2\}$, let $\mathcal{L}_i \subseteq \Sigma_i^*$ be a language and $\mathcal{L}_{\mathcal{S}_i} \subseteq \mathcal{L}_i$ be its corresponding secret language. Suppose that the shared alphabet $\Sigma_s \subseteq \Sigma_a \subseteq \Sigma$. If $\mathcal{L}_{\mathcal{S}_i}$ is opaque with respect to \mathcal{L}_i and $\Sigma_{a,i}$, then $\mathcal{L}_{\mathcal{S}_i}$ is opaque with respect to $\mathcal{L}_1 || \mathcal{L}_2$ and Σ_a .

Proof. Let $t \in \mathcal{L}_{\mathcal{S}_1} || \mathcal{L}_2$, $t_1 = P_{\Sigma_1}(t) \in \mathcal{L}_1$, and $t_2 = P_{\Sigma_2}(t) \in \mathcal{L}_2$. Since $\mathcal{L}_{\mathcal{S}_1}$ is opaque wrt \mathcal{L}_1 and $\Sigma_{a,1}$, by Definition 4, $\exists t'_1 \in \mathcal{L}_1 \setminus \mathcal{L}_{\mathcal{S}_1}$ such that $P_{\Sigma_{a,1}}(t_1) = P_{\Sigma_{a,1}}(t'_1)$. By Lemma 17, $\exists t' \in \{t'_1\} || \{t_2\}$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$. Thus, $t' \in \mathcal{L}_1 || \mathcal{L}_2$ by Remark 2. So, $(\forall t \in \mathcal{L}_{\mathcal{S}_1} || \mathcal{L}_2) \exists t' \in (\mathcal{L}_1 || \mathcal{L}_2) \setminus (\mathcal{L}_{\mathcal{S}_1} || \mathcal{L}_2)$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$. Thus, $\mathcal{L}_{\mathcal{S}_1}$ is opaque with respect to $\mathcal{L}_1 || \mathcal{L}_2$ and Σ_a . This generalizes for $\mathcal{L}_{\mathcal{S}_i}$, where $1 \leq i \leq n$. ■

Corollary 19. Consider LTSs G_1, \dots, G_n with secret state sets \mathcal{S}_i for $1 \leq i \leq n$. Let $\Sigma_s \subseteq \Sigma_a$. If \mathcal{S}_i is opaque with respect to G_i and $\Sigma_{a,i}$, then \mathcal{S}_i is opaque with respect to $G_1 || \dots || G_n$ and Σ_a .

Corollary 20. If $\forall i \in \{1, \dots, n\}$, \mathcal{S}_i is opaque w.r.t. $G_1 || \dots || G_n$ and Σ_a , then $\mathcal{S} = \bigcap_{i=1}^n \mathcal{S}_i = \mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1} \times \mathcal{S}_i \times \mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n$ is opaque w.r.t. $G_1 || \dots || G_n$ and Σ_a .

Because opacity is preserved by composition when $\Sigma_s \subseteq \Sigma_a$, we can introduce another optimization, similar to the techniques used by Saboori and Hadjicostis (2010). Rather than building the composed system at the start of the algorithm, we can verify that \mathcal{S}_i is opaque with respect to $G_1 || \dots || G_n$ and Σ_a by incrementally checking if it is opaque for larger and larger compositions, starting with the local module G_i . If, at any point, \mathcal{S}_i is opaque with respect to a composition's result and Σ_a , it must be opaque with respect to the whole system $G_1 || \dots || G_n$ and Σ_a by Theorem 18. This is formalized in Algorithm 2.

This method allows us to verify opacity for each module's secret \mathcal{S}_i much more efficiently. In the worst case, we still need to build $Det_{\Sigma_{a,1}}(G_1) || \dots || Det_{\Sigma_{a,n}}(G_n)$ to verify opacity of each \mathcal{S}_i . However, for systems where all modules are already opaque individually, we only need to construct

Algorithm 2 Verify opacity of \mathcal{S}_i with respect to $G_1 || \dots || G_n$ and Σ_a .

input: G_i and its secret states \mathcal{S}_i , all LTSs G_1, \dots, G_n , and the attacker alphabets $\Sigma_{a,j}$ for $j \in \{1, \dots, n\}$.

Assumes $\Sigma_s \subseteq \Sigma_a$ (pairwise shared events must be observed by the attackers).

output: true iff \mathcal{S}_i is opaque wrt $G_1 || \dots || G_n$ and Σ_a .

```

1: if  $\mathcal{S}_i$  is opaque wrt  $Det_{\Sigma_{a,i}}(G_i)$  and  $\Sigma_a$  then
2:   return true ▷ By Corollary 19.
3:  $\mathcal{D} \leftarrow Det_{\Sigma_{a,i}}(G_i)$ 
4: for all  $G_j \in \{G_1, \dots, G_n\} \setminus \{G_i\}$  do
5:    $\mathcal{D} \leftarrow \mathcal{D} || Det_{\Sigma_{a,j}}(G_j)$ 
6:   if  $\mathcal{S}_i$  is opaque with respect to  $\mathcal{D}$  and  $\Sigma_a$  then
7:     return true ▷ By Corollary 19.
8: return false

```

$Det_{\Sigma_{a,i}}(G_i)$ for each module, allowing us to verify opacity without building the monolithic system.

Theorem 21. Algorithm 2 is in $\Theta(2^{mn})$.

Proof. Let $m = \max_{i \in \{1, \dots, n\}} |\mathcal{Q}_i|$, the maximum size of a module's state set. In the worst case, we multiply n structures of size 2^m and we check each of these states to verify if it is opaque. Thus, the algorithm is $\Theta(2^{mn})$. ■

Example 22. Let us consider again the determinized LTS of Figure 2, we observe that neither $\mathcal{S}_1 = \{3\}$ nor $\mathcal{S}_2 = \{3'\}$ are opaque with respect to their respective modules and Σ_a . Thus, we must compose $Det_{\Sigma_{a,1}}(G_1) || Det_{\Sigma_{a,2}}(G_2)$ to verify whether each \mathcal{S}_i is opaque. The result is in Figure 5. From this, we can observe that \mathcal{S}_1 is not opaque with respect to $G_1 || G_2$ and Σ_a , but \mathcal{S}_2 is because $\{3'\}$ is no longer accessible. □

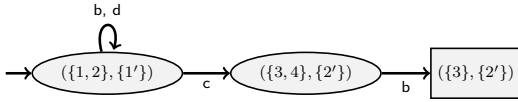


Fig. 5. $Det_{\Sigma_{a,1}}(G_1) || Det_{\Sigma_{a,2}}(G_2)$ with $\Sigma_s = \{c\}$ and $\Sigma_a = \{b, c, d\}$

3.3 Heuristics for Opacity Verification

Algorithm 2 is run for every secret \mathcal{S}_i . This worst case complexity only occurs in interconnected systems (i.e., with one connected component in the interaction graph) where \mathcal{S}_i is not opaque. This is considerably better than taking the product of all modules and determinizing the result, which would be $\mathcal{O}(2^{m^n})$. Nevertheless, it would be desirable to check for some secrets simultaneously. In particular, if we add G_j to the composition, we can check if \mathcal{S}_j is opaque in the composition at the same time as checking \mathcal{S}_i . This means we do not need to run the algorithm separately for \mathcal{S}_j . This has the potential to reduce the number of times the algorithm must be run.

The runtime of Algorithm 2 depends on the selection of which LTS to compose first. Composing G_i with one LTS might immediately make \mathcal{S}_i opaque with respect to the composed system and Σ_a , but with another, it might not. Thus, while correctness does not depend on the ordering, algorithm runtime does. Some heuristics to note include composing LTSs in the order of the most shared events or

the fewest local events. Heuristics for choosing the order of LTSs in incremental verification algorithms are discussed in Brandin et al. (2004).

4. CONTROLLER SYNTHESIS

In cases where secrets are not opaque, we can use supervisory control to enforce opacity. In particular, we are interested in generating controllers locally for every module to enforce the local module's secret. Leveraging the fact that opacity holds over composition when we assume $\Sigma_s \subseteq \Sigma_a$, we present an algorithm for generating more permissive controllers than those produced by standard algorithms.

4.1 Problem Formulation

We are interested in synthesizing n controllers $\mathcal{C}_1, \dots, \mathcal{C}_n$. Each controller \mathcal{C}_i is local to a single G_i and enforces the opacity of \mathcal{S}_i with respect to the controlled monolithic system and Σ_a . That is, while the local controller \mathcal{C}_i knows the construction of the whole system, it is only concerned about keeping the local secret \mathcal{S}_i and it can only observe and control local events in module G_i .

Globally, we have an observable alphabet $\Sigma_o \subseteq \Sigma$ and controllable alphabet $\Sigma_c \subseteq \Sigma_o$. Since each controller is local to its own module, \mathcal{C}_i has the observable alphabet $\Sigma_{o,i} = \Sigma_o \cap \Sigma_i$ and the controllable alphabet $\Sigma_{c,i} = \Sigma_c \cap \Sigma_i \subseteq \Sigma_{o,i}$. As in the previous section, we let $\Sigma_{a,i} = \Sigma_a \cap \Sigma_i$.

Problem 23. Suppose we have n modules G_i with secrets \mathcal{S}_i and a global attacker view \mathcal{A} with alphabet Σ_a . Further suppose that we want to build local controllers \mathcal{C}_i that have an observable alphabet $\Sigma_{o,i}$ and a controllable alphabet $\Sigma_{c,i}$ such that $\Sigma_{c,i} \subseteq \Sigma_{o,i} \subseteq \Sigma_i$, and each controller enforces the opacity of only one secret \mathcal{S}_i . How can we generate a controller \mathcal{C}_i for every G_i such that $\mathcal{S} = \bigcup_{i=1}^n (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque w.r.t. $(\mathcal{C}_1/G_1) || \dots || (\mathcal{C}_n/G_n)$ and Σ_a ?

4.2 Enforcing Opacity using Control

First, we want to ensure that if we perform control to enforce the opacity of one secret, we do not expose another secret.

Theorem 24. Let $\Sigma_s \subseteq \Sigma_a$. If \mathcal{S}_1 is opaque with respect to \mathcal{C}_1/G_1 and $\Sigma_{a,1}$, then for any valid controller \mathcal{C}_2 for G_2 , \mathcal{S}_1 is opaque with respect to $(\mathcal{C}_1/G_1) || (\mathcal{C}_2/G_2)$ and Σ_a .

Proof. By Theorem 18 and Corollary 19, opacity is preserved by composition when $\Sigma_s \subseteq \Sigma_a$. Thus, regardless of what we compose with \mathcal{C}_1/G_1 , \mathcal{S}_1 is opaque with respect to the composition and Σ_a . ■

This leads us to a simple, but naive, approach for synthesizing controllers to enforce opacity of \mathcal{S}_i with respect to the system $G_1 || \dots || G_n$ and Σ_a .

Proposition 25. To enforce the opacity of \mathcal{S}_i with respect to the system $G_1 || \dots || G_n$ and Σ_a , we can get the supremal controller for enforcing opacity in the local module. By doing this for every module, we find that \mathcal{S} is opaque with respect to $(\mathcal{C}_1/G_1) || \dots || (\mathcal{C}_n/G_n)$ and Σ_a .

Proof. We know that \mathcal{S}_i is opaque with respect to \mathcal{C}_i/G_i and $\Sigma_{a,i}$ by definition of the supremal controller. By Theorem 24, we know that every \mathcal{S}_i is opaque with respect to $(\mathcal{C}_1/G_1) \parallel \dots \parallel (\mathcal{C}_n/G_n)$ and Σ_a . By Corollary 13, we know $\mathcal{S} = \bigcup_{i=1}^n (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque with respect to $(\mathcal{C}_1/G_1) \parallel \dots \parallel (\mathcal{C}_n/G_n)$ and Σ_a . ■

Unfortunately, this approach can be restrictive, resulting in less permissive controllers than we would have if we considered information about other modules as well.

Example 26. Consider G_2 in Figure 2. Suppose that $\Sigma_{o,2} = \Sigma_2$ and $\Sigma_{c,2} = \{\mathbf{d}\}$ (i.e., \mathbf{c} is uncontrollable). Since $\{3'\}$ is a secret state, we need to disable the controllable transitions leading to it. Because \mathbf{c} is uncontrollable, we can observe that no valid supremal controller can exist. However, we have already shown that \mathcal{S}_2 is opaque with respect to $G_1 \parallel G_2$ and Σ_a . In the global system, no control is needed to enforce opacity of \mathcal{S}_2 . Thus, using the naive method to produce individual controllers is restrictive. □

One way to make the controllers more permissive is to use a fully permissive controller whenever \mathcal{S}_i is opaque with respect to $G_1 \parallel \dots \parallel G_n$. To ensure that this works, we need to show that performing control on other modules will not affect the opacity of \mathcal{S}_i .

Theorem 27. Let $\mathcal{L}_1 \subseteq \Sigma_1$, $\mathcal{L}_2 \subseteq \Sigma_2$, and $\Sigma_s \subseteq \Sigma_a$. If $\mathcal{L}_{\mathcal{S}_1} \subseteq \mathcal{L}_1$ is opaque with respect to $\mathcal{L}_1 \parallel \mathcal{L}_2$ and Σ_a , then for any $\mathcal{L}'_2 \subseteq \mathcal{L}_2$, $\mathcal{L}_{\mathcal{S}_1}$ is opaque with respect to $\mathcal{L}_1 \parallel \mathcal{L}'_2$.

Proof. Consider all $t \in \mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}_2$. Let $t_1 = P_{\Sigma_1}(t) \in \mathcal{L}_1$ and $t_2 = P_{\Sigma_2}(t) \in \mathcal{L}_2$. Since $\mathcal{L}_{\mathcal{S}_1}$ is opaque with respect to $\mathcal{L}_1 \parallel \mathcal{L}_2$ and $\Sigma_{a,1}$, by Definition 4, $\exists t' \in (\mathcal{L}_1 \parallel \mathcal{L}_2) \setminus (\mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}_2)$ such that $P_{\Sigma_a}(t) = P_{\Sigma_a}(t')$. So, we know there is a $t'_1 = P_{\Sigma_1}(t') \in \mathcal{L}_1 \setminus \mathcal{L}_{\mathcal{S}_1}$. There are two cases we need to consider. In the first case, suppose $t_2 \in \mathcal{L}'_2$. Then, $t \in \mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}'_2$ and $t' \in (\mathcal{L}_1 \parallel \mathcal{L}'_2) \setminus (\mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}'_2)$ by Remark 1. In the second case, suppose $t_2 \notin \mathcal{L}'_2$. Then, $t \notin \mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}'_2$ and $t' \notin (\mathcal{L}_1 \parallel \mathcal{L}'_2) \setminus (\mathcal{L}_{\mathcal{S}_1} \parallel \mathcal{L}'_2)$ by Remark 1. Thus, opacity holds: $\mathcal{L}_{\mathcal{S}_1}$ is opaque with respect to $\mathcal{L}_1 \parallel \mathcal{L}'_2$. ■

This means that restricting behavior of other modules will not affect the opacity of the current module's secret, allowing us to synthesize more permissive controllers. In the algorithm below, \mathcal{C}^\dagger represents the supremal controller generated by the algorithm from Tong et al. (2018).

Algorithm 3 Synthesizing controller \mathcal{C}_i for module G_i .

input: G_i and its secret states \mathcal{S}_i , all LTSs G_1, \dots, G_n , the attacker alphabets $\Sigma_{a,j}$ for $j \in \{1, \dots, n\}$, and the observable and controllable alphabets $\Sigma_{o,i}$ and $\Sigma_{c,i}$.

output: \mathcal{C}_i , the controller for G_i .

- 1: **if** \mathcal{S}_i is opaque wrt $G_1 \parallel \dots \parallel G_n$ and Σ_a **then**
 - 2: **return** a fully permissive controller
 - 3: $G \leftarrow G_1 \parallel \dots \parallel G_n$
 - 4: $\mathcal{S} \leftarrow (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$
 - 5: **return** \mathcal{C}^\dagger enforcing opacity of \mathcal{S} wrt G , Σ_a , $\Sigma_{o,i}$, and $\Sigma_{c,i}$
-

The controller \mathcal{C}^\dagger synthesized by Algorithm 3 has observable alphabet $\Sigma_{o,i}$ and controllable alphabet $\Sigma_{c,i}$; even though the controller knows the construction of all modules, it cannot necessarily observe all of the events.

Furthermore, at it observes only $\Sigma_{o,i}$ and cannot disable events outside $\Sigma_{c,i}$, it is a controller \mathcal{C}_i for module G_i . We compute \mathcal{C}_i from Algorithm 3 in two steps: we first check if opacity holds, in which case a fully permissive controller suffices. Otherwise, we compute the supremal controller \mathcal{C}^\dagger that enforces opacity of the module's secret \mathcal{S}_i , w.r.t. the monolithic system $G_1 \parallel \dots \parallel G_n$ and attacker's alphabet Σ_a , using the algorithm from Tong et al. (2018). We consider the entire system to build the controller rather than a single module G_i , as it results in more permissive controllers.

By Remark 15, it suffices to only consider modules in the same connected component in the interaction graph (such as the graph in Figure 4): enforcing opacity in the connected component will enforce it for the global system. In Algorithm 3, we assume that G_1, \dots, G_n are all part of the same connected component in the interaction graph. The algorithm would still work if they were not in the same connected component, but with reduced efficiency.

Theorem 28. If we use Algorithm 3 to synthesize every \mathcal{C}_i , then $\mathcal{S} = \bigcup_{i=1}^n (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ is opaque with respect to $(\mathcal{C}_1/G_1) \parallel \dots \parallel (\mathcal{C}_n/G_n)$ and Σ_a .

Proof. By construction of the supremal controller, \mathcal{S} is opaque w.r.t. $(\mathcal{C}_i/G_i) \parallel (G_{j_1} \parallel \dots \parallel G_{j_m})$ and Σ_a . ■

Recall that we assumed that all modules are in the same connected component. By Remark 15, even if more modules in a different connected component are added, opacity still holds.

Note that the controller \mathcal{C}_i is observable and controllable by construction of the supremal controller. When there are other connected components, controllability still holds with the full system because controllability holds under arbitrary parallel composition.

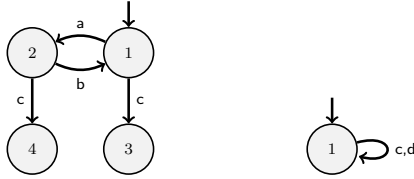
There are two standard assumptions when synthesizing a supremal optimal controller: either the attacker knows the construction of both the system and the controller, or it only knows the system. For the first case, to compute such a controller, Dubreil et al. (2010) assume that $\Sigma_a \subseteq \Sigma_o$ or $\Sigma_o \subseteq \Sigma_a$. In general, neither of these hold. The observable alphabet for the controller is $\Sigma_{o,i}$ and the attacker's alphabet is Σ_a ; that is, the attacker can observe events from other modules. Thus, in many cases, there does not yet exist an algorithm for finding a supremal controller when the attacker knows the architecture of both the system and the controller. For the second case, an algorithm from Tong et al. (2018) only requires that the controllable alphabet is a subset of the observable alphabet. Our controllers satisfy this property because $\Sigma_{c,i} \subseteq \Sigma_{o,i}$. Thus, we focus on this latter case.

Theorem 29. Algorithm 3 is in $\mathcal{O}(2^{2(m^n \times 2^{m^n} + |\Sigma_c|)})$.

Proof. Let $m = \max_{i \in \{1, \dots, n\}} |\mathcal{Q}_i|$, the maximum size of a module's state set. Building $G_1 \parallel \dots \parallel G_n = (\mathcal{Q}, \Sigma, \delta, q_0)$ requires assembling (at most) n structures of size m , so $|\mathcal{Q}|$ is smaller than m^n . We then generate the supremal controller of this structure using the algorithm of Tong et al. (2018), which is $\mathcal{O}(2^{2(|\mathcal{Q}| \times 2^{|\mathcal{Q}|} + |\Sigma_c|)})$. Thus, Algo-

rithm 3 synthesizes a controller for one module, and is in $\mathcal{O}(2^{2(m^n \times 2^{m^n} + |\Sigma_c|)})$ ■

Example 30. Consider again the example of Figure 1. Since we know \mathcal{S}_2 is opaque with respect to $G_1||G_2$ and Σ_a from Figure 5, that means that \mathcal{C}_2 can be the fully permissive controller (we assume $\Sigma_{o,2} = \Sigma_2$). To enforce opacity of \mathcal{S}_1 , we have to restrict the behavior of G_1 using the supremal controller. Assuming that $\Sigma_{o,1} = \Sigma_1$ and $\Sigma_{c,1} = \{a, b\}$, the final controller \mathcal{C}_1 is depicted in Figure 6. Note that this is more permissive than the naive approach from Proposition 25: in this case, the naive approach does not yield a set of valid controllers. □



(a) \mathcal{C}_1 , the controller for G_1 (b) \mathcal{C}_2 , the controller for G_2

Fig. 6. Modular controllers produced using Algorithm 3 to enforce the opacity of \mathcal{S}_1 and \mathcal{S}_1 w.r.t. $G_1||G_2$ and Σ_a

We can now prove that synthesizing opacity-preserving controllers for each connected component of the interaction graph separately gives a set of controllers that preserve all secrets.

Theorem 31. Consider a global supremal controller \mathcal{C}^\dagger that enforces the opacity of $\mathcal{S} = \bigcup_{i=1..n} (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$ with respect to $G_1||\dots||G_n$ and Σ_a . Then, if $\Sigma_s = \emptyset$, $\mathcal{L}((\mathcal{C}_1/G_1)||\dots||(\mathcal{C}_n/G_n)) = \mathcal{L}(\mathcal{C}^\dagger/(G_1||\dots||G_n))$.

Proof. Let $\mathcal{S}'_i = (\mathcal{Q}_1 \times \dots \times \mathcal{Q}_{i-1}) \times \mathcal{S}_i \times (\mathcal{Q}_{i+1} \times \dots \times \mathcal{Q}_n)$. As $\Sigma_s = \emptyset$, every module is a separate connected component and Algo. 3 only considers the local module G_i when synthesizing \mathcal{C}_i (recall that the Algo 3 runs only on modules in the same connected component).

Suppose $t_1.\sigma.t_2 \in \mathcal{L}_{\mathcal{S}_i}(G_i)$, with $\sigma \in \Sigma_{c,i}$ and $t_2 \in (\Sigma_i \setminus \Sigma_{c,i})^*$. Suppose there is no non-secret string $t' \in \mathcal{L}(G_i) \setminus \mathcal{L}_{\mathcal{S}_i}(G_i)$ where $P_{\Sigma_a}(t_1.\sigma.t_2) = P_{\Sigma_a}(t')$. Then, \mathcal{C}_i must disable σ after observing $P_{\Sigma_{o,i}}(t_1)$. Now, consider all $t'_1 \in \mathcal{L}(G_1||\dots||G_n)$ where $P_{\Sigma_i}(t'_1) = t_1$. Since $\Sigma_s = \emptyset$, we know that $t'_1.\sigma.t_2 \in \mathcal{L}_{\mathcal{S}'_i}(G_1||\dots||G_n)$ and there is no corresponding non-secret string. Thus, we know that \mathcal{C} must disable σ after seeing $P_{\Sigma_o}(t'_1)$, as well, because $t_2 \in (\Sigma \setminus \Sigma_c)^*$. Thus, anything \mathcal{C}_i disables is also disabled by \mathcal{C} : $\mathcal{L}(\mathcal{C}/(G_1||\dots||G_n)) \subseteq \mathcal{L}((\mathcal{C}_1/G_1)||\dots||(\mathcal{C}_n/G_n))$. But since $\mathcal{L}((\mathcal{C}_1/G_1)||\dots||(\mathcal{C}_n/G_n))$ is controllable and observable, and because \mathcal{S} is opaque w.r.t. this controlled system and Σ_a , we know $\mathcal{L}((\mathcal{C}_1/G_1)||\dots||(\mathcal{C}_n/G_n)) \subseteq \mathcal{L}(\mathcal{C}/(G_1||\dots||G_n))$; the supremal global controller is maximally permissive by definition. Thus, their languages are equal. ■

5. CONCLUSIONS

Under the assumption that the attacker can observe the interface between modules, we have presented a strategy for validating and ensuring that every set of secret states \mathcal{S}_i

is opaque with respect to a composed system $G_1||\dots||G_n$ and Σ_a by composing modules only as needed. This makes it possible to evaluate opacity in large modular systems, where traditional algorithms might not be feasible since they involve constructing the monolithic system. Furthermore, our algorithm for generating local controllers makes it possible to enforce opacity for each module separately. This is an essential technique when a controller cannot effectively control modules—for instance, if they are in different geographical locations.

To extend this research, we are interested in synthesizing controllers where the attacker is aware of the controllers' structure by extending techniques from Dubreil et al. (2010). We are also interested in how this could be extended to modal transition systems, another model which provides some freedom of implementation of the modules.

REFERENCES

- Brandin, B., Malik, R., and Malik, P. (2004). Incremental verification and synthesis of discrete-event systems guided by counter-examples. *IEEE Trans Contr Syst Technol*, 12(3), 387–401.
- Bryans, J.W., Koutny, M., Mazaré, L., and Ryan, P.Y.A. (2008). Opacity generalised to transition systems. *Int J Inf Secur*, 7(6), 421–435.
- Cassez, F., Dubreil, J., and Marchand, H. (2012). Synthesis of opaque systems with static and dynamic masks. *Form Methods Syst Des*, 40(1), 88–115.
- Contant, O., Lafortune, S., and Teneketzis, D. (2006). Diagnosability of discrete event systems with modular structure. *Discrete Event Dyn Syst*, 16(1), 9–37.
- de Queiroz, M. and Cury, J. (2000). Modular control of composed systems. In *ACC*, 4051–4055. IEEE.
- Dubreil, J., Darondeau, P., and Marchand, H. (2008). Opacity enforcing control synthesis. In *WODES*, 28–35.
- Dubreil, J., Darondeau, P., and Marchand, H. (2010). Supervisory control for opacity. *IEEE Trans Automat Contr*, 55(5), 1089–1100.
- Jacob, R., Lesage, J.J., and Faure, J.M. (2016). Overview of discrete event systems opacity. *ARC*, 41, 135–146.
- Jezequel, L. and Fabre, E. (2012). Turbo planning. In *WODES*, 301–306.
- Lin, F. and Wonham, W. (1988). On observability of discrete-event systems. *Info Sci*, 44(3), 173 – 198.
- Mazaré, L. (2005). Decidability of opacity with non-atomic keys. In *FAST*, 71–84.
- Ramadge, P.J. and Wonham, W.M. (1982). Supervision of discrete event processes. In *CDC*, 1228–1229.
- Saboori, A. and Hadjicostis, C.N. (2010). Reduced-complexity verification for initial-state opacity in modular discrete event systems. In *WODES*, 78–83.
- Tong, Y., Li, Z., Seatzu, C., and Giua, A. (2018). Current-state opacity enforcement in discrete event systems under incomparable observations. *Discrete Event Dyn Syst*, 28(2), 161–182.